Fast, Best-Effort Real-Time Scheduling Algorithms

Abstract

This paper presents two fast, best-effort real-time scheduling algorithms called MDASA and MLBESA. MDASA and MLBESA are novel in the way that they heuristically, yet accurately mimic the behavior of the DASA and LBESA scheduling algorithms, but are faster with $O(n)$ and $O(n \log(n))$ worst-case complexities, respectively. Experimental results show that the performance of MDASA and MLBESA perform, in general, is close to that of DASA and LBESA, respectively, for a broad range of realistic workloads. However, for highly bursty workload, MLBESA is found to perform worse than LBESA. Furthermore, the task response times under MDASA and MLBESA are very close to the values under their counterpart scheduling algorithms. Thus, MDASA and MLBESA can substitute DASA and LBESA algorithms, respectively, in adaptive resource allocation techniques for asynchronous real-time distributed systems where DASA and LBESA have previously been serious bottlenecks on computational costs.

Index Terms

Best-Effort Real-Time Scheduling, Overload Scheduling, Response Time Analysis, Asynchronous Real-Time Systems, Distributed Real-Time Systems

I. Introduction

Asynchronous real-time distributed systems are emerging in many domains including defense, telecommunication, and industrial automation for the purpose of strategic mission management [24]. Such systems are fundamentally distinguished by the significant run-time uncertainties that are inherent in their application environment and system resource states [22], [24], [25]. Consequently, it is difficult to postulate upper bounds on application workloads for such systems that will always be respected at run-time.

An example of such asynchronous real-time distributed systems is US Navy's Anti-Air Warfare (AAW) system. AAW is an example air-defense system that is envisioned to be onboard US Navy's future surface combatants. To better understand AAW, a software prototype that approximated AAW called *DynBench*, was developed as part of DARPA's Quorum program [25]. *DynBench* was first discussed in [46], later in [47], and in detail in [39]. Such an asynchronous real-time distributed system is often operated in environments that are subject

to significant uncertainties—parameters such as event arrival rates cannot be accurately characterized at design time [16], [22].

Given the aforementioned uncertainties, the classical hard real-time satisfiability objective of "always meet all timing constraints" and hard real-time scheduling algorithms that achieve the satisfiability objective are not cost-effective or even impossible for asynchronous real-time distributed systems. Thus, in this paper, we consider timeliness requirements that are described by Jensen's benefit accrual model [22]. This model generalizes the hard real-time "deadline" timing constraint to encompass non-binary timing constraints. That is, completion of a task at anytime yields an arbitrary benefit that is specified by the task's *benefit function*. For example, a benefit function may specify a constant benefit before a task's deadline time and zero benefit after that, which is also known as "firm deadline" in real-time literatures and is our model used in this paper.

To deal with the inherent non-determinism's in asynchronous real-time distributed systems, we have developed adaptive resource allocation algorithms such as the RBA* (Response-Time Best-Effort Resource Allocation) algorithm [20]. The RBA* algorithm seeks to maximize the aggregate benefit of trans-node real-time computations that have end-to-end timeliness requirements, specified using benefit functions. Furthermore, it considers "best-effort" process scheduling algorithms that have the scheduling objective of maximizing accrued benefit on a single end-host.

These scheduling algorithms are called "best-effort" in the sense that they seek to provide the best benefit to the application tasks, where the best benefit that can be accrued by an application task is application-specified using benefit functions. More precisely, the scheduling objective is to maximize Accrued Benefit Ratio (ABR)—the ratio of total accrued benefit to the sum of all task benefits.

Two of the most prominent best-effort real-time scheduling algorithms include the Dependent Activity Scheduling Algorithm (DASA) algorithm [17] and Locke's Best Effort Scheduling Algorithm (LBESA) [30]. DASA and LBESA are equivalent to the Earliest Deadline First (EDF) algorithm during under-loaded conditions [17], where EDF is optimal and guarantees that all deadlines are always satisfied. In the event of an overload situation, DASA and LBESA seek to maximize the aggregate task benefit.

DASA and LBESA are known to outperform other single processor real-time scheduling algorithms (during overload situations) in general. For example, in [17], Clark shows that

DASA generally outperforms LBESA during overload situations. In addition, in [20], Hegazy shows that DASA outperforms the Robust Earliest Deadline First (RED) [14] and the Robust High Density (RHD) [13] scheduling algorithms. Furthermore, variants of DASA and LBESA have been implemented in the Alpha real-time operating system [23] and the MK7.3 kernel [42]. There are other implementations of overload real-time scheduling algorithms, such as the Maximum Urgency First (MUF) scheduler [41] implemented inside the CHIMERA II real-time operating system. However, to the authors' best knowledge, DASA and LBESA are the only two benefit accrual scheduling algorithms that have been implemented and deployed in mission-critical systems [16].

In spite of these remarkable results of DASA and LBESA, they have been serious performance bottlenecks for resource allocation algorithms such as RBA*. The RBA* algorithm is a proactive algorithm in the sense that it allocates resources to tasks for a future time window. To do that, RBA* needs to determine task response times for the future time window, e.g., the RBA* algorithm is invoked every several seconds. However, due to the complexity of analyzing process response times under DASA, RBA* incurs a worst-case computational complexity that is a polynomial of the 4th degree. [1] In fact, in a middleware implementation of the RBA* algorithm [28], we have observed that the overhead of invoking RBA* can be up to 1 sec, which is considerably high for deadlines in the order of a few seconds.

Analyzing process response times under best-effort scheduling algorithms is difficult, because best-effort schedulers such as DASA and LBESA make decisions at each scheduling event that are functions of the remaining process execution times at the event. Thus, RBA* estimates response times by determining the scheduling events that occur during a time interval (under which the response times are to be determined) and by applying the scheduling algorithm at each scheduling event to determine the scheduling decision. This is computationally expensive, as we incur the cost of the scheduler—$O\left(n^2\right)$ for DASA and LBESA given n tasks in the ready queue [17], [30]—whenever the scheduler is invoked at a scheduling event.

Thus, we are motivated to develop fast best-effort scheduling algorithms that can yield the same performance as that of DASA and LBESA, if not better. Furthermore, if such new scheduling algorithms behave similar to DASA and LBESA, they can then be used by resource allocation algorithms such as RBA* for determining response times under DASA

[1] Given n trans-node tasks, a maximum of m processes per task, p processors, a minimum task period of k, and an adaptation window of length W, RBA* incurs a worst-case computational complexity of $O\left(p^2 m^4 n^4 \lceil W/k \rceil^4\right)$.

and LBESA, respectively, in a faster way. This will reduce the computational complexity of RBA*, resulting in good performance at low computational cost.

In this paper, we present two best-effort scheduling algorithms called Modified DASA (or MDASA) and Modified LBESA (or MLBESA) that exactly addresses the aforementioned problem. MDASA and MLBESA are novel in the way that they heuristically, yet accurately, mimic the behavior of the DASA and LBESA algorithms, but are faster with $O(n)$ and $O(n \log(n))$ worst-case complexities, respectively. MDASA and MLBESA reason about the behavior of DASA and LBESA by heuristically determining a feasible schedule of the process ready queue.

Our experimental results show that in general, MDASA and MLBESA perform almost as good as DASA and LBESA, respectively. However, under highly bursty and heavily overloaded situations, DASA and LBESA may outperform MDASA and MLBESA. Furthermore, the process response times under MDASA and MLBESA are also found to be very close to the values under their counterpart scheduling algorithms. While MDASA has better performance than MLBESA and has better worst-case complexity, MLBESA guarantees the optimal schedule during underload situations.

The rest of the paper is organized as follows: We discuss the task model considered by MDASA and MLBESA in Section II. Section III and Section IV describe MDASA and MLBESA, respectively. We present the experimental evaluation of the algorithms in Section V. In Section VI, we discuss the past and related works and contrast them with MDASA and MLBESA. Finally, the paper concludes with a summary of the work, its contributions, and identify future work in Section VII.

II. The Task Model

We consider a set of real-time tasks (or processes)[2] that are executed on a single processor. The set of tasks is denoted by the set $T = \{T_1, T_2, \ldots, T_n\}$. Further, all tasks are assumed to be preemptable, and therefore can be preempted at any time during their execution. Furthermore, tasks do not share any non-CPU resources or have precedence relations with other tasks, and therefore are independent of each other.

Each task T_i is characterized by an arrival time A_i and an execution time C_i. For simplicity, we assume that the task execution time C_i is known by the scheduler at the time of arrival.

[2] We will use the terms *task* and *process* interchangeably in the paper unless otherwise stated.

This can be achieved by application profiling techniques i.e., deriving the relationship between task workload and execution time, as done in [1], [37]. Similar to [17], we assume "rectangular" benefit functions for all tasks.Thus, completing a task anytime before its deadline will result in uniform benefit; completing it after the deadline will result in zero benefit.

We denote the benefit of a task T_i—the height of T_i's benefit function—as B_i and the absolute deadline of a task T_i as d_i, respectively. [3]

We use $R_i(t)$ to denote the remaining execution time of a task T_i at time t. It is easy to see that $R_i(t) \leq C_i, \forall t \geq 0$. The slack of task T_i at time t is therefore defined as $S_i(t) = d_i - t - R_i(t), \forall t \geq 0$.

For completeness, we also include definitions of several quantities and terms here:

1) For a generic real-time system with n tasks, where tasks can be activated dynamically i.e., all tasks are aperiodic tasks, the processor load during the time interval of $[t, d_i]$ is given by $\rho_i(t) = \frac{\sum_{d_k \leq d_i} c_k(t)}{(d_i - t)}$ ($i \in [1, ..., n]$), where $c_k(t)$ refers to the remaining execution time of task T_k with deadline earlier or equal to d_i [14]. Further, the total load in the interval of $[t, d_n]$ is defined as $\rho = max_i(\rho_i)$ [14]. Furthermore, if $\rho \leq 1$, we say that the system is under-loaded. Otherwise, we say the system is overloaded [14];
2) A schedule is an order of assigning tasks to the processor. That is, a schedule contains ordered executions of tasks;
3) We say a schedule is *feasible* if it can satisfy all timing constraints, such as deadlines of tasks within the schedule. Otherwise, the schedule is *infeasible*. A set of tasks is feasible if and only if there exits a feasible schedule on the set of tasks. If the task set is under-loaded, EDF can produce a feasible schedule on the task set due to EDF's optimality [18]; and
4) Competitive factor measures the worst case performance of an algorithm. This concept was introduced in [5], and is defined as follows: A scheduling algorithm A has a competitive factor φ_A if and only if it can guarantee a cumulative value (or benefit) $\Gamma_A \geq \varphi_A \Gamma^*$, where Γ^* is the cumulative value (or benefit) achieved by the optimal clairvoyant scheduler.

[3] The "*benefit*" of a task is also called "*value*" or "*utility*" in the literature [10].

III. The MDASA Scheduling Algorithm

The DASA algorithm makes scheduling decisions using the concept of *benefit densities*. The benefit density of a task is the benefit accrued per unit time by the execution of the task. Thus, benefit density defines a measure of the "return of investment" for the task. We denote the benefit density of a task T_i at time t as $BD_i(t)$, which is given by $BD_i(t) = B_i/R_i(t)$.

The objective of DASA is to compute a schedule Γ that will maximize the aggregate task benefit. Aggregate task benefit is the cumulative sum of the benefit accrued by the execution of the tasks. Thus, a schedule that satisfies all deadlines of all tasks will clearly yield the maximum aggregate benefit (since task benefit functions are step-benefit functions).

During under-load conditions, we know that EDF produces such a feasible schedule for all tasks [18]. Such an all–task–feasible schedule Γ will contain all tasks in T, if all tasks are present in the task ready queue. However, if all task deadlines cannot be satisfied, then the schedule that will yield the maximum benefit will be an exact subset of the task set i.e., $\Gamma \subset T$, since some tasks will have to be excluded from the schedule. Therefore, DASA always seeks to compute a feasible schedule within which tasks can accrue as much benefit as possible.

Thus, at each scheduling event, DASA examines tasks in the task ready queue in decreasing order of their benefit densities. The algorithm then inserts each task into a tentative schedule at its deadline-position and checks the feasibility of the schedule. Tasks are maintained in increasing deadline-order in the tentative schedule. If inserting a task into the tentative schedule results in an infeasible schedule, then the algorithm removes the task from the schedule.

DASA repeats this process until all tasks in the ready queue have been examined. The algorithm then selects the earliest deadline task in the tentative schedule (which will be at the "head" of the schedule) as the next task to be executed. Note that if all task deadlines can be satisfied, then DASA's output will be the same as that of EDF.

The rationale behind DASA's examination of tasks in decreasing order of their benefit densities is that if tasks are examined in decreasing order of their benefit densities, then at any instant in time, the task that is included in the schedule can accrue the maximal benefit per time unit among the set of non-examined tasks. This will increase the chance of collecting as much benefit per time unit as possible, thereby increasing the likelihood of maximizing aggregate benefit.

Given n tasks in the task ready queue, DASA incurs a worst-case complexity of $O(n^2)$ at each scheduling event. The computationally intensive steps of the algorithm include (1) sorting the tasks to order them in decreasing order of their benefit densities, which costs $O(n log(n))$, and (2) testing for feasibility of the tentative schedule each time a task is inserted into the schedule.

The feasibility-testing costs $O(n^2)$. This is because, to determine the feasibility of a schedule, each task deadline must be examined in the increasing order of task deadlines. Each task deadline must then be compared against the cumulative sum of the remaining execution times of all tasks with lesser deadlines than the task deadline. This means that to determine the feasibility of the schedule, DASA makes an $O(n)$ pass over the task list. Since the algorithm tests for feasibility of the tentative schedule every time a task is inserted into the schedule, the resulting cost becomes $O(n^2)$.

Thus, the bottleneck of DASA's scheduling cost is the repeated feasibility test, which costs $O(n)$ for a single test. Thus, we believe that one possible way to speed up the algorithm is to determine the feasible task subset without the repeated feasibility test.

A. Observations and Heuristic

Our basic observation is that the tasks in the ready queue fall into one of the three classes:

- **Class I**: tasks that will never appear in Γ;
- **Class II**: tasks that will definitely be in Γ; and
- **Class III**: tasks that may appear in Γ, i.e., those that do not belong to Class I or Class II.

The MDASA algorithm identifies tasks in Class I and Class II by determining if inserting a task T_i into Γ can maintain the feasibility of Γ. In general, most of the tasks in ready queue can be identified as either Class I or Class II task and hence decreases the possibility of making a different scheduling decision from that of DASA.

If a task T_i belongs to Class III, then it is impossible to determine whether the feasibility of Γ can be maintained without performing a feasibility test. In this case, MDASA *heuristically* calculates the probability that T_i will be included in Γ. We denote this probability as P_i.

Before describing MDASA, we first formulate the observations that identify the tasks in Class I and Class II as follows.

Observation III.1. *A task T_i cannot be in the feasible schedule Γ if $S_i(t) < 0$.*

Proof: It follows that T_i will fail the feasibility test. Thus, only tasks with non-negative slack could be in the feasible schedule. □

Observation III.2. *Let T_f be the first task that has a non-negative slack in decreasing benefit density order among all tasks. Then T_f must be in the feasible schedule Γ.*

Proof: Note that the DASA algorithm examines the task ready queue in decreasing order of benefit densities. Thus, at the point of examining the task T_f, the tentative schedule Γ (which is feasible) is initialized to be empty and no task has been successfully inserted. Therefore, the expanded schedule $\Gamma \cup T_f = T_f$ should be a feasible schedule as well. □

Now that T_f is in the feasible schedule Γ, any other task T_j that would cause task T_f to miss its deadline should be eliminated from Γ, as stated in Observation III.3.

Observation III.3. *If task T_j satisfies $d_j < d_f$ and $R_j(t) > S_f(t)$, then inserting T_j into Γ will cause the T_f to miss its deadline.*

Proof: Since $d_j < d_f$, T_j will be inserted before T_f in Γ. Hence, the start time of T_f will be postponed by time $R_j(t)$. This postponement implies that T_f will miss its deadline because $R_j(t) > S_f(t)$. □

Observation III.4. *If a T_k that has a non-negative slack satisfies $d_k > d_{max}$, where d_{max} is the latest deadline in Γ, then T_k can be in Γ iff*

$$\frac{\sum_{T_i \in \Gamma} R_i(t) + R_k(t)}{d_k - t} \leq 1.0 \qquad (1)$$

Proof: Recall that the feasible schedule Γ is ordered by increasing task absolute deadlines. Thus, task T_k will be inserted at the *tail* of Γ, which cannot affect the feasibility of any subset of Γ. Therefore, the feasibility of task set $T_k \cup \Gamma$ only depends upon if the accumulative processor time demand until d_k exceeds the available time $(d_k - t)$. This condition becomes Equation 1.

In practice, the scheduling algorithm may keep track of the total remaining execution time of all tasks currently in Γ, i.e. $\sum_{T_i \in \Gamma} R_i(t)$, which results in constant complexity of using Equation 1. Also notice that this feasibility condition is sufficient and necessary. Thus, the task T_k defined as above must belong to either Class I or Class II. □

Once tasks that belong to Class I and Class II are identified, we can now proceed to the remaining tasks that constitute Class III.

Recall that in DASA, tasks are inserted into the tentative schedule at their deadline positions [17]. That is, tasks in the tentative schedule are deadline-ordered. If the insertion of a new task T_i at its deadline position will cause any of the tasks that are already in schedule Γ to miss its deadline, T_i should not be inserted into Γ. This procedure is called "feasibility test" that ensures the feasibility of the tentative schedule. Therefore, for a task T_i in Class III, whether or not it should be included in Γ depends upon how significantly the insertion of T_i will affect the feasibility of Γ.

Furthermore, we observe that early deadline tasks are inserted near the beginning of the schedule by DASA. Our working hypothesis is that tasks with earlier deadlines are more likely to interfere with other tasks, causing them to miss their deadlines. Although the success of our resulting algorithms suggests this to be true (see Section V), we have not examined any other heuristics.

We denote the number of tasks in the current Γ that have later deadlines than d_i as k, which can be approximated by the following equation:

$$k = |\Gamma| \times \frac{d_{max} - d_i}{d_{max} - d_{min}} \tag{2}$$

where d_{max} and d_{min} are the latest and earliest deadlines among all tasks currently in Γ, respectively. Thus, k is the approximated number of tasks that will be affected by inserting T_i.

At this stage, it is not clear how P_i can be calculated based on the task parameters and k. Though there are a number of possible methods to calculate P_i, MDASA adopts a straightforward way, where it computes P_i as $P_i = \frac{1}{k}$. Our motivation for using this simple heuristic is to minimize the overhead of the scheduler. Another advantage is that P_i is automatically normalized, i.e., P_i is guaranteed to be within the range of $(0, 1]$ since $k \geq 1$.[4]

Note that the k value computed by Equation 2 may not be a good approximation for task queues with non-uniformly distributed deadlines. For example, if majority of the tasks have deadlines close to the earliest deadline and only a few tasks have very late deadlines, the calculation of k using Equation 2 tends to overestimate the impact of the insertion of a new task. Consequently, the MDASA algorithm may overly reject tasks and thus may result in poor performance. We show performance of the MDASA algorithms under various experimental

[4] If the calculated k is less than 1, it should be rounded to 1, because it is the number of tasks to be affected.

scenarios in Section V.

B. Description of the MDASA Algorithm

Note that MDASA only schedules independent tasks. Thus, it is triggered at two scheduling events: (1) arrival of a new task, and (2) termination of the currently executing task. Other scheduling events of the original DASA algorithm, namely *request resource* and *release resource*, are not needed by MDASA.

Algorithm III.1 MDASA_schedule

```
1: Input: Benefit density ordered task ready queue BQ;
2: Initialize Γ := φ, C := 0, T_r := φ; /* T_r is the selected task to execute */
3: for each task T_i ∈ BQ in descending order of BD_i do
4:   if S_i(t) > 0 then
5:     if Γ = φ then
6:       /* T_f is the first non-negative slack task */
7:       Γ := Γ ∪ T_i, d_max := d_min := d_i, T_f := T_i;
8:     if d_i > d_max then
9:       if (C + R_i(t)) / (d_i − t) ≤ 1.0 then
10:        Γ := Γ ∪ T_i, d_max := d_i, C := C + R_i(t);
11:    else
12:      if d_i < d_f then
13:        if R_i(t) < S_f(t) /* S_f(t) is the slack of task T_f */ then
14:          ProbInclude(Γ, T_i);
15:      else
16:        ProbInclude(Γ, T_i);
17: return T_r;
```

Algorithm III.2 ProbInclude

```
1: Input: Γ and associated state variables d_min, C, and T_r; task T_i;
2: P_i := (d_max − d_min)/(|Γ| × (d_max − d_i));
3: if rand() < P_i /* rand() returns a random value within [0,1]*/ then
4:   Γ := Γ ∪ T_i; C := C + R_i(t);
5:   if d_min > d_i then
6:     d_min := d_i; T_r := T_i; /* T_r is the selected task */
```

To facilitate the computation of scheduling decisions, MDASA maintains the task ready queue in the order of non-decreasing task benefit densities. When a new task arrives, MDASA inserts the task into the ready queue at its benefit density position. If the processor is not idle, the algorithm extracts the currently executing task from the ready queue, updates its remaining execution time and its benefit density, and then re-inserts the task into the ready queue at its new benefit density position. To select the next task to execute, MDASA invokes the algorithm *MDASA_schedule* (Algorithm III.1).

In the event of a task termination, MDASA removes the currently executing task from the ready queue. The algorithm then invokes *MDASA_schedule* to select the next task to execute.

TABLE I
TASK READY QUEUE SCHEDULED BY MDASA AT $t = 10$

Tasks	B_i	$BD_i(t)$	d_i	$d_i - t$	$R_i(t)$	$S_i(t)$	Class	in Γ
T_1	360	60	15	5	6	-1	I	No
$T_2(T_f)$	250	50	20	10	5	5	II	Yes
T_3	80	40	14	4	2	2	III	Yes
T_4	180	30	18	8	6	2	I	No
T_5	200	20	22	12	10	2	I	No
T_6	20	10	30	20	2	18	II	Yes

As shown in Algorithm III.1, MDASA accepts a (benefit density-ordered) task ready queue as input and selects the next task to be executed. The algorithm examines the tasks in the ready queue in decreasing order of their benefit densities. By Observation III.1, only tasks with non-negative slacks can be in the feasible schedule. Thus, MDASA silently discards all tasks with negative slacks.

If a task T_i with a non-negative slack has the latest deadline, Equation 1 from Section III-A is used to check if the insertion of T_i can maintain the feasibility of Γ. Otherwise, its deadline is compared with the deadline of task T_f i.e., d_f.

Recall that T_f is the first non-negative slack task in decreasing order of benefit density. By Observation III.3, any task T_i that will cause task T_f to miss its deadline should be eliminated from Γ. Recall that any task that does not belong to Class I or Class II falls into Class III. Thus, tasks in Class III are identified in the procedure of identifying Class I and Class II tasks in the task ready queue. These Class III tasks are further examined by the *ProbInclude* algorithm (Algorithm III.2), which computes the probability P_i and determines if T_i should be included in Γ. Though a single scheduling decision in Algorithm III.2 depends on the output of the $rand()$ function and is thus random, this randomness is amortized if the algorithm is invoked repeatedly for a large number of times. Similar randomized scheduling policy has been employed in algorithms such as Lottery Scheduling [45].

Note that the latest and earliest task deadlines in Γ are computed in the procedure for computing the P_i's. Also, the task T_r is selected whenever a task T_i is determined in Γ and has the earliest deadline. Therefore, the MDASA algorithm, unlike the original DASA algorithm, only needs one pass and does not invoke the repeated feasibility test.

We now show an example of how MDASA algorithm makes the scheduling decision. Assume that the MDASA scheduling algorithm is triggered at $t = 10$. The triggering event could be an arrival of a new task or the termination of the currently executing task. In either case,

the resulting task ready queue (benefit density ordered) is shown in Table I.

Task T_1 is eliminated from Γ due to its negative slack time. Task T_2 has positive slack time and is the first non-negative slack task in descending order of benefit density. Thus, it becomes T_f and is included in Γ.

Task T_3 has a positive slack and an earlier deadline than T_f. By the *MDASA_Schedule* algorithm, its remaining execution time is compared against the slack time of T_2. Since $R_3(10) < S_2(10)$, task T_3 belongs to Class III. Whether T_3 should be included in Γ depends upon the output of $ProbInclude(\Gamma, T_3)$. In this example, assume T_3 is included in Γ.

Task T_4 has earlier deadline than T_2 and satisfies $R_4(10) > S_2(10)$. Therefore, the insertion of T_4 will cause T_2 to miss its deadline. Task T_5 has the latest deadline but fails the test by Equation 1. On the contrary, task T_6 has the latest deadline and enough slack i.e., $\frac{C + R_6(10)}{d_6 - 10} \leq 1.0$, where $C = R_2(10) + R_3(10) = 7$. The selected task T_r is the earliest deadline task in Γ, which is T_3 in this example.

C. Worst-Case Complexity of MDASA

To analyze the worst-case complexity of MDASA, we consider a maximum of n tasks.

If the scheduler is triggered by a task arrival, the insertion of the new task involves searching for the correct task position in the schedule. This is followed by an insert operation. Using a binary search, the worst-case complexity of these operations is given by $O(\log(n)) + O(n) = O(n)$.

In the event of a non-idle processor, the state variables of the currently running task T_r need to be updated and T_r must be replaced in the ready queue to maintain the benefit density order. The task T_r can be kept track of by saving the index of T_r in a variable. Thus, the replacement operation of T_r could be implemented by searching the correct position for the updated BD_r and a "swap" operation. This costs $O(\log(n))$.

The MDASA scheduler then invokes the MDASA_schedule algorithm to make a scheduling decision. Algorithm *MDASA_schedule* examines the task ready queue from the highest benefit density task to the lowest benefit density task. The worst-case complexity of *MDASA_schedule* is $O(n)$ as *MDASA_schedule* makes only a single pass over the ready queue.

Therefore, the total worst-case complexity of MDASA at a scheduling event is given by $O(n) + O(\log(n)) + O(n) = O(n)$.

IV. The MLBESA Algorithm

LBESA [30] is another best-effort real-time scheduling algorithm. It is similar to DASA in that both algorithms schedule tasks using the notion of benefit densities, and are equivalent to EDF during under-load situations. However, the algorithms differ in the way they reject tasks during overload situations. In [17], Clark shows that DASA is generally better than LBESA in terms of aggregate accrued task benefit.

While DASA examines tasks in the ready queue in decreasing order of their benefit densities for determining feasibility, LBESA examines tasks in the increasing order of task deadlines. Like DASA, LBESA also inserts each task into a tentative schedule at its deadline-position and checks the feasibility of the schedule. Tasks are maintained in increasing deadline-order in the tentative schedule. If the insertion of a task into the tentative schedule results in an infeasible schedule, then unlike DASA, LBESA removes the *least benefit density task* from the tentative schedule. LBESA continuously removes the least benefit density task from the tentative schedule until the tentative schedule becomes feasible. Once all tasks in the ready queue have been examined and a feasible tentative schedule is thus constructed, LBESA selects the earliest deadline task from the tentative schedule.

Again, the worst-case complexity of LBESA is $O(n^2)$, given n tasks in the ready queue. Furthermore, the major component of the $O(n^2)$ complexity is the repeated feasibility test performed by the algorithm before rejecting each task.

To speed up LBESA, our approach is to maintain the order of rejecting tasks and to *heuristically* determine the tasks to be rejected without invoking the feasibility test. We now discuss our observations, the resulting MLBESA heuristic, and its worst-case complexity in the subsections that follow.

A. Observations and Heuristic

Note that LBESA continuously rejects tasks in increasing benefit density order if there is an overload. This means that if $BD_i > BD_j$ and task T_i is rejected, then task T_j must also have been rejected.

For convenience, we denote the set of tasks that must be rejected to produce a feasible task subset as V. Therefore, to determine the tasks in V, we only need to determine the size of V.

Without loss of generality, we assume that the tasks are deadline-ordered in the set $T =$

$\{T_1, T_2, \ldots, T_n\}$. Furthermore, recall that a *feasible* task set satisfies $\rho \leq 1$, where ρ is the load on the task set [14]. Figure 1 plots the load of a set of five example tasks specified in Table II.

TABLE II
A SET OF 5 TASKS

Tasks	$R_i(t)$	$d_i - t$	$\rho_i(t)$
T_1	4	5	0.8
T_2	5	6	1.5
T_3	3	10	1.2
T_4	17	20	1.45
T_5	1	25	1.2

TABLE III
REMOVING T_1 FROM THE TASK SET

Tasks	$R_i(t)$	$d_i - t$	$\rho_i(t)$
T_2	5	6	0.83
T_3	3	10	0.8
T_4	17	20	1.25
T_5	1	25	1.04

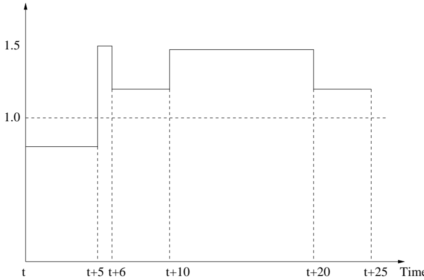

Fig. 1. Example Load Calculation

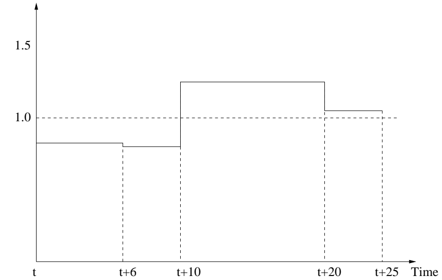

Fig. 2. Load After Removing T_1

Our first observation on the load definition is formulated as the following:

Observation IV.1. *Let $\rho_m(t)$ be the load calculated during the time interval $[t, d_m]$ such that it is the maximum load among all $\rho_i(t)$ of the task set. Then, rejecting one or more tasks such that $\rho_m(t) \leq 1.0$ is necessary, but not sufficient, to ensure the feasibility of the remaining tasks.*

Proof: The necessity of this observation is obvious. We use the task set specified in Table II to clarify that it is not sufficient. Assume that task T_1 is rejected due to its minimum benefit density. Before T_1 is rejected, $\rho_m = \rho_2 = 1.5$. After T_1 is removed from the task set, ρ_2 decreases to 0.83, as shown in Table III. However, ρ_4 and ρ_5 are still larger than 1.0 and hence the remaining task set is not feasible. Figure 2 plots the processor load after removing task T_1. □

Therefore, we can partition the task subset V into two parts: $V = V_1 \cup V_2$, where V_1 is the task subset that must be rejected to decrease ρ_m to be less than 1.0, and V_2 is the task

subset that must be rejected to ensure the feasibility of the remaining tasks.

To determine the decrease in load that is acquired by rejecting one task, we use the following observation.

Observation IV.2. *After rejecting task T_j, the processor load calculated during interval $[t, d_i]$ is given by:*

$$\rho'_i = \begin{cases} \rho_i - \dfrac{R_j(t)}{d_i - t} & \text{if } d_j \le d_i \\ \rho_i & \text{otherwise} \end{cases}, \tag{3}$$

where ρ_i and ρ'_i are the calculated load before and after the task rejection, respectively.

Proof: If $d_j \le d_i$, then the total processor time demand that needs to be completed before d_i must be subtracted from $R_j(t)$ in the event of removing task T_j. It follows

$$\rho'_i = \frac{\sum_{d_k \le d_i} R_k(t) - R_j}{d_i - t} = \rho_i - \frac{R_j(t)}{d_i - t}.$$

In the case of $d_j > d_i$, the processor time demand of task T_j is not included in ρ_i. Thus, removing T_j cannot affect ρ_i. □

Therefore, a natural way to identify and reject tasks in the subset V_1 is to continuously reject tasks, from the lowest benefit density task to the highest benefit density task, until ρ_m is less than 1.0. After each task is rejected, ρ_m is re-computed using Equation 3.

Since rejecting tasks in V_1 is necessary, but not sufficient, to ensure the feasibility of the remaining tasks, we *heuristically* determine and reject another task subset $V_2 = V - V_1$. Observe that a high ρ_m means that there is a severe overload situation for the task set. Thus, it is reasonable to use ρ_m to heuristically determine how many tasks should be rejected. These tasks are rejected to resolve the overload situation, as a higher ρ_m generally implies that more tasks should be rejected (though not necessarily).

We define the term *Reject Ratio* as the ratio of the number of tasks to be rejected from the subset $(T - V_1)$.

Definition IV.1. The reject ratio of subset $(T - V_1)$ is defined as follows:

$$\eta = \frac{|V_2|}{|T - V_1|}. \tag{4}$$

Note that determining the exact number of tasks in V_2 involves performing the feasibility test, which is not desired. Thus, we approximate the exact reject ratio with the help of ρ_m.

This approximated reject ratio, indicated as η', is computed as follows:

$$\eta' = \frac{\rho_m - 1}{\rho_m} = 1 - \frac{1}{\rho_m} \leq 1. \tag{5}$$

Now that the reject ratio is computed, we can determine the size of the subset $|V_2|$ by definition IV.1.

$$|V_2| = \eta' \times |T - V_1| = \eta' \times (|T| - |V_1|). \tag{6}$$

We choose this simple heuristic due to its implementation efficiency and its property of automatic normalization. It is worth noting that under certain workload scenarios, this heuristic may not perform well. For example, if a task set is only slightly overloaded and the deadline order of a task queue is in the reserve order of its benefit density order, i.e., later deadline tasks have lower benefit densities, LBESA may still need to reject most of the tasks whereas MLEBSA only rejects a small number of tasks due to a small η'. We examine the performance of the MLBESA algorithm in Section V.

Once the size of V_1 and V_2 are determined, the algorithm simply rejects $(|V_1| + |V_2|)$ tasks, from the lowest benefit density task to the highest benefit density task. The remaining tasks are conjectured to be feasible without performing any feasibility test.

Note that if ρ_m is extremely large, the approximated reject ratio η' will be close to 1. In this case, all tasks should be rejected, which implies that no feasible schedule exists. Then the algorithm executes the earliest deadline task from the remaining tasks, if there is any. In the example 5-task set, $|V_1| = 1$, because rejecting task T_1 is enough to reduce ρ_2 to 0.83. Furthermore, the reject ratio is approximated as $\eta' = 1 - \frac{1}{\rho_2} = 0.33$. Therefore, $|V_2| = \eta' \times (|T| - |V_1|) = 0.33 \times 4 = 1.33$.

B. Description of the MLBESA Algorithm

The MLBESA scheduling algorithm is also triggered by task arrival and task termination events. However, unlike the MDASA algorithm, MLBESA orders the task ready queue in increasing deadlines to facilitate the initial feasibility test.

When a new task arrives, MLBESA inserts the task into the task ready queue at its deadline position, and then invokes the Algorithm *MLBESA_schedule* (Algorithm IV.1) for determining the next task to execute. If the algorithm is triggered by the termination of a task, MLBESA removes the currently running task from the task ready queue and then invokes the Algorithm *MLBESA_schedule*.

Algorithm IV.1 MLBESA_schedule

```
1: Input: Absolute deadline ordered task ready queue DQ = {T_1, T_2, ..., T_n};
2: Initialize internal variables: ρ_m := 0; C := 0; T_r := φ; |V_1| := |V_2| := 0;
3: t := get_current_time();
4: for each task T_i ∈ DQ in non-decreasing order of deadlines do
5:     C := C + R_i(t); /* C is the total processor time demand */
6:     ρ_i(t) := C / (d_i − t);
7:     if ρ_i(t) > ρ_m then
8:         ρ_m := ρ_i(t); d_m := d_i;
9: if ρ_m ≤ 1.0 then
10:    return T_r := T_1;
11: else
12:    Sort DQ in another queue BQ, by benefit densities;
13:    η' := 1 − 1/ρ_m;
14:    for each task T_i ∈ BQ in ascending order of benefit densities do
15:        Remove task T_i from BQ, BQ := BQ − T_i; |V_1| := |V_1| + 1;
16:        if d_i ≤ d_m then
17:            ρ_m := ρ_m − R_i(t) / (d_m − t);
18:        if ρ_m ≤ 1.0 then
19:            |V_2| := η' × (|DQ| − |V_1|);
20:            break;
21:    for i := 1 to |V_2| do
22:        Remove the lowest benefit density task from BQ;
23: if BQ ≠ φ then
24:    Select T_r ∈ BQ, which has the earliest deadline among all tasks within BQ;
25: return T_r;
```

The *MLBESA_schedule* algorithm first checks the feasibility of the task set by examining the tasks (in the ready queue) in their deadline-order. The algorithm identifies the maximum load factor ρ_m and its associated deadline d_m. If no overload is detected i.e., $\rho_m \leq 1$, then the algorithm simply applies the EDF algorithm to select the next task to be executed, which is task T_1 at the "head" of the deadline-ordered queue.

In the event of an overload, the algorithm first computes the reject ratio η'. Tasks are then continuously rejected in ascending order of their benefit densities until $\rho_m \leq 1$. The number of tasks rejected so far is the size of the subset V_1. Once $|V_1|$ tasks are rejected, *MLBESA_schedule* determines the size of the subset $|V_2|$ using Equation 6 from Section IV-A. Note that the tasks in $V_1 \cup V_2$ are *temporarily* removed from the task ready queue. Such tasks may be determined to be feasible at future scheduling events. After $|V_1| + |V_2|$ tasks have been rejected, the algorithm selects the earliest deadline task from the remaining task set, if there exists any.

C. Worst-Case Complexity of MLBESA

To analyze the complexity of MLBESA, we consider a deadline-ordered ready queue, denoted as DQ. We store DQ as a linear array. Thus, inserting a new task involves searching

for the correct deadline position of the task in DQ, followed by the insert operation. Given n tasks in DQ, this has a worst-case complexity of $O(\log(n)) + O(n) = O(n)$.

MLBESA first checks the feasibility of the task set by making a single pass over DQ, which costs $O(n)$. If no overload is detected, the algorithm returns the earliest deadline task in DQ i.e., the task at the head of DQ. In this case, the complexity of MLBESA is $O(n)$.

However, in the event of an overload, the task queue needs to be reordered according to benefit densities. Tasks in subset $V_1 \cup V_2$ are then continuously rejected by the algorithm to produce a feasible subset. In the worst-case, all tasks in the ready queue could be rejected. Thus, the complexity of MLBESA during overload is the sum of the cost of sorting the task ready queue (according to benefit densities) and rejecting the tasks. This total cost becomes $O(n \log(n)) + O(n) = O(n \log(n))$.

Clearly, the worst-case complexity of MLBESA is dominated by the complexity during the overload situation. Therefore, the worst-case complexity of MLBESA is $O(n \log(n))$.

It is interesting to note that the algorithm could also order the task ready queue according to benefit densities, as in the case of the MDASA algorithm. This benefit density ordered-ready queue will help to reduce the complexity of rejecting the tasks during an overload situation, but will incur the same total worst-case complexity. Furthermore, such a benefit density-ordered task ready queue must be reordered according to deadlines. Our approach of ordering the task ready queue by deadlines may avoid the expensive sorting operation if there is no overload. Therefore, the deadline order scheme has a better average-case complexity.

V. Performance Evaluation

In experimentally evaluating MDASA and MLBESA, our main goal is to determine how well the algorithms perform with respect to their counterpart DASA and LBESA scheduling algorithms, respectively. The performance of MDASA and MLBESA are also compared against the D^{over} algorithm [26] and the BE-v algorithm proposed in [34]. A detailed discussion of related algorithms can be found in Section VI. Furthermore, we are interested in determining the task response times under MDASA and MLBESA. We conducted simulation studies to determine these performance metrics using synthetic workloads.

A. Performance of MDASA and MLBESA

To evaluate the performance of the scheduling algorithms, we considered tasks with randomly distributed parameters. Each experimental setting was characterized by four param-

eters including task execution time C_i, slack S_i, timeliness benefit B_i, and task inter-arrival time I_i.

Table IV and Table V summarize the baseline experimental settings for exponential distributions and normal distributions, respectively. Note that the task slack is specified as the ratio of its slack time to its corresponding execution time. Once C_i and S_i of a task T_i are determined, the relative deadline of the task is given by $D_i = C_i + S_i$. Furthermore, the task inter-arrival time depends on the average load ρ_A. [5] Thus, only the ratio $\frac{I_i}{C_i}$ is specified.

TABLE IV
EXPONENTIAL TASK PARAMETERS

Para	Distribution	Mean	Std. Dev.
C_i	exponential	0.5 sec	0.5 sec
$\frac{S_i}{C_i}$	exponential	0.25	0.25
B_i	exponential	10	10
$\frac{I_i}{C_i}$	exponential	$\frac{1.0}{\rho}$	$\frac{1.0}{\rho}$

TABLE V
NORMAL TASK PARAMETERS

Para	Distribution	Mean	Std. Dev.
C_i	normal	0.5 sec	0.5 sec
$\frac{S_i}{C_i}$	normal	0.25	0.25
B_i	normal	10	10
$\frac{I_i}{C_i}$	normal	$\frac{1.0}{\rho}$	$\frac{1.0}{\rho}$

Each simulation experiment lasted 8,000 seconds and generated a stream of tasks, based on the specified task parameters. We conducted experiments for $\rho_A = 0.1 \sim 2.0$. The number of tasks generated during each experiment varied from approximately 1,000 to 43,000, due to the change in the average load. To eliminate random effects, each experiment was independently repeated five times for the same task stream.

We measured the *Deadline Satisfaction Ratio (DSR)* and *Accrued Benefit Ratio (ABR)* produced by MDASA and MLBESA. DSR is defined as the ratio of the number of satisfied deadlines to the total number of tasks. Figure 3 and Figure 4 show the *DSR* and *ABR* of MDASA and that of DASA under two baseline experimental settings. As we can see, MDASA performs almost exactly the same as DASA for both experimental settings. The difference in the *DSR* and *ABR* of the two algorithms is found to be generally less than 5%. This result implies that the MDASA algorithm almost always makes the same scheduling decisions as that of DASA.

To evaluate the performance of the scheduling algorithms for highly bursty workload, we also considered the general Pareto distribution for task parameters. The general Pareto distribution is interesting because its variance could be infinity given a finite mean value—

[5] This average processor load definition has been used in real-time literature such as [13], [17], which may or may not be directly related to the processor load defined in Section II.

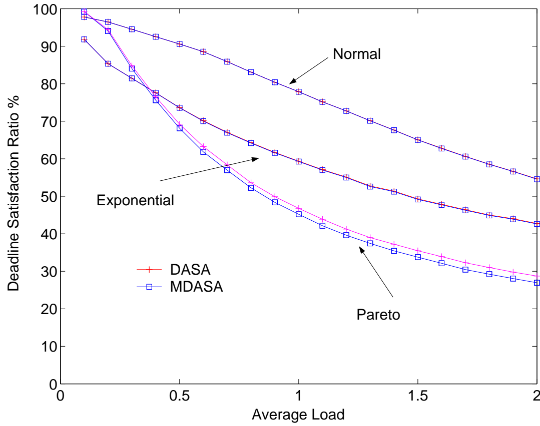

Fig. 3. DSR's of MDASA and DASA

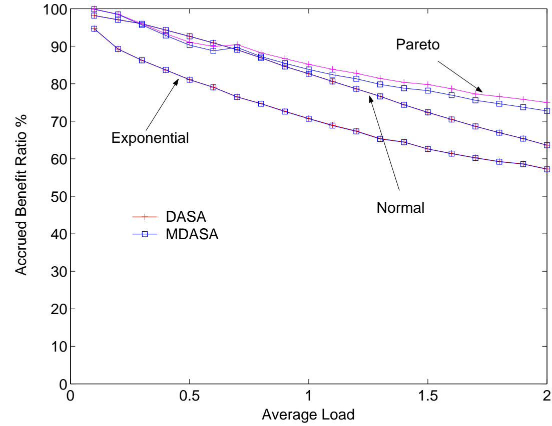

Fig. 4. ABR's of MDASA and DASA

the maximal non-determinism one can have. [6] Furthermore, the general Pareto distribution is the base for synthesizing self-similar communication traffic that can be found in real-time applications such as compressed video transmission (see [27] for an introduction to self-similar traffic).

In our baseline Pareto distribution experiments, task execution times, slack times, inter arrival times, and task benefit values all follow the general Pareto distribution. For task execution time, the average task execution time is 0.5 sec and the α value is 1.9. Note that once the average value and the α value are determined, β of the distribution can be calculated as $Mean \times (\alpha - 1)/\alpha$. Similarly, the average slack time is five times of the task execution time—a long slack time that allows tasks remain feasible for a long time and the average task benefit is 10. Furthermore, the α values for the distributions of slack time, task benefit, and task benefit are all set to 1.1.

We show DSR's and ABR's of MDASA and DASA schedulers under the general Pareto distribution in Figure 3 and Figure 4, respectively. Compared with the performance under exponential and normal distributions, both DASA and MDASA satisfies fewer percentages of task deadlines. Furthermore, DASA slightly outperforms MDASA in terms of DSR. This performance gap is echoed in Figure 4, where DASA also performs better than MDASA. In addition, we observe that both DASA and MDASA accrue more benefit under the general Pareto distribution than under normal or exponential. This is because the general Pareto

[6] A general Pareto distribution is governed by a shape parameter α and a scale parameter β that determines the minimal value of the random variable. If α is between 1 and 2, the variance of the distribution is infinity and the mean value is $\beta\alpha/(\alpha - 1)$ [44].

distribution may produce very large task benefit values for a few tasks. Thus, satisfying deadlines of these high-benefit tasks can dramatically increase the algorithm performance in terms of ABR.

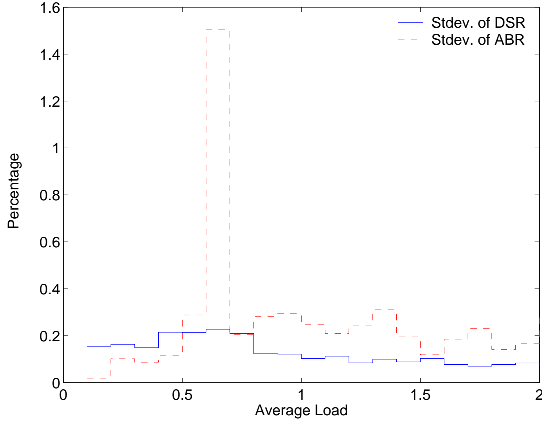

Fig. 5. Standard Deviations of MDASA Performance

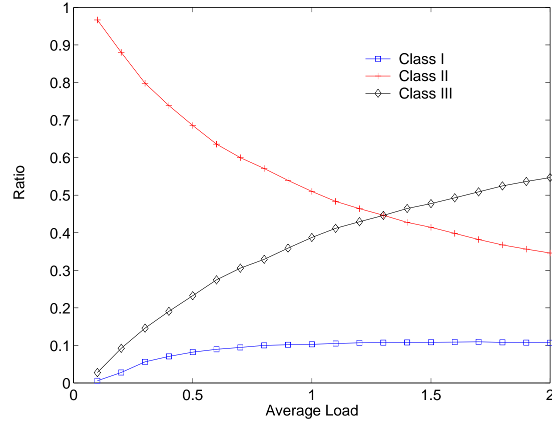

Fig. 6. Percentages of Tasks in MDASA Classes

Since MDASA uses the *rand*() function to heuristically compute the schedule, it is possible that random factors can affect the algorithm performance given a set of of input tasks. Thus, for a given input task stream, we repeat the experiments with five independent random seeds that are used by the MDASA algorithm. These five experiments are grouped together as they all have the same input. Standard deviations of DSR's and ABR's of the MDASA scheduler are then calculated to evaluate the effects of random factors.

We show the average standard performance deviations of MDASA for five groups of experiments in Figure 5 (using the base-line Pareto distributions). As shown in the figure, performance of MDASA is reasonably stable i.e., the performance standard deviations are no more than 2% in our experiments. Therefore, we conclude that the performance of MDASA is close to that of DASA for a broad range of workload scenarios.

Furthermore, Figure 6 shows the ratios of tasks that fall into each class of the MDASA algorithm under the base line general Pareto distribution. We observe that the ratio of Class I tasks i.e., at any given scheduling event, the number of Class I tasks to the total number of tasks in the ready queue remains approximately 0.1 when load is greater than 0.6.

On the contrary, the ratio of Class II tasks monotonically decreases while the ratio of Class III tasks continues increasing. This is because heavy load tends to generate long ready queues and more tasks can not be identified as Class I or Class II. Furthermore, the increasing ratio of Class III tasks explains why MDASA performs worse than DASA during heavy load.

We also observe that even when the system is heavily overloaded i.e., load is 2.0 and the workload pattern is highly bursty i.e., task parameters follow the general Pareto distribution, approximately half of the tasks still fall into either Class I or Class II.

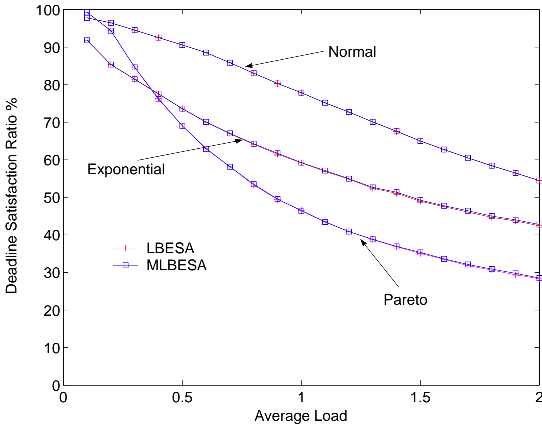

Fig. 7. DSR's of MLBESA and LBESA

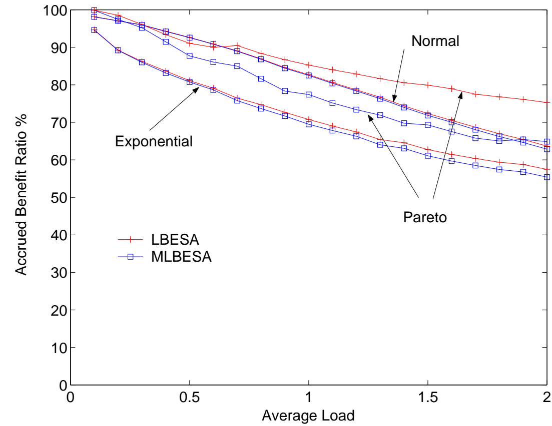

Fig. 8. ABR's of MLBESA and LBESA

We conducted the same experiments for MLBESA and LBESA. The DSR's and ABR's of MLBESA versus LBESA are shown in Figure 7 and Figure 8. We observe that under normal and exponential distributions, MLBESA is only slightly worse than LBESA. However, under the general Pareto distribution, LBESA may accrue as much as 10% more benefit than MLBESA, which is considered as a visible performance gap. We conjecture that this is because under highly bursty workload pattern, MLBESA may reject too many tasks that have high benefit but long execution time as well. Thus, these rejected tasks have non-negligible impacts on the aggregate benefit ratio.

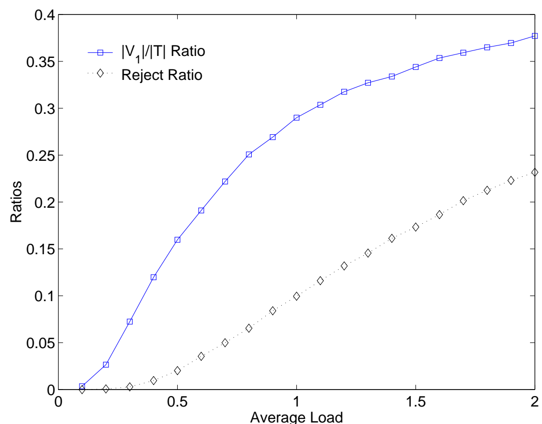

Fig. 9. Percentages of Tasks in MLBESA Categories

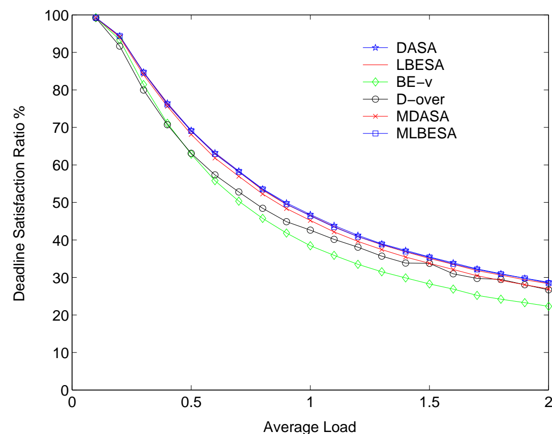

Fig. 10. Comparison of DSR's (Pareto Distribution)

Furthermore, in Figure 9, we show the average ratio of tasks that belong to MLBESA task set V_1 and the average reject ratio computed by MLBESA. As can be seen from the figure, both ratios increase when load increases. It is also worth to note that no more than 35% percentage of all tasks (at load 2.0) belong to task set V_1 that are discarded to possibly produce feasible schedules. Recall that the MLBESA algorithm heuristically determines if a task in the remaining task set $(T - V_1)$ is in the schedule. Thus, a relatively small $|V_1|/T$ ratio (compared with MDASA's Class I and Class II ratios in Figure 6) implies good chances that MLBESA makes different scheduling decisions from LBESA. This result explains the performance gap between LBESA and MLBESA in Figure 8.

As a comparison, we implemented the D^{over} algorithm [26] and the BE-v algorithm [34] in the simulation experiments.

The D^{over} algorithm has been proved to be optimal in the sense that it achieves the upper bound of the competitive factor for any on-line scheduling algorithm. Note that the competitive factor of an on-line algorithm measures its *worst-case* performance. However, the optimal competitive factor of D^{over} does not imply the best *average* performance, because D^{over} may make pessimistic scheduling decisions such that it can guarantee the optimal competitive factor.

The BE-v algorithm was inspired by the D^{over} algorithm. Furthermore, Mosse et al. show that BE-v outperforms the original LBESA algorithm dealing with short teaser tasks.

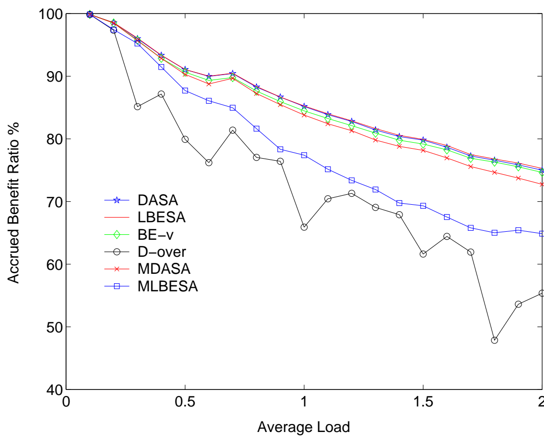

Fig. 11. Comparison of ABR's (Pareto Distribution)

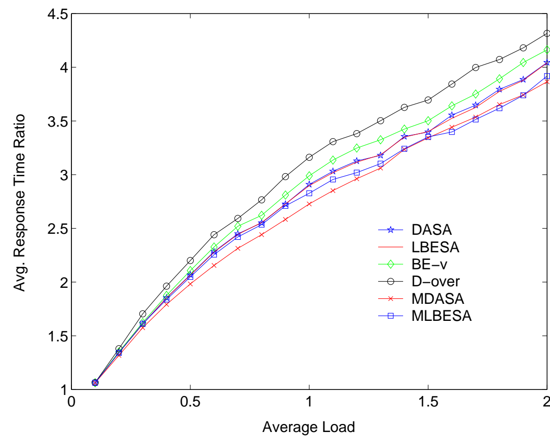

Fig. 12. Comparison of Response Time Ratios

We first show the DSR's and ABR's of the six algorithms under the base line general Pareto distributions. Our reason for choosing the general Pareto distribution is because a highly

busrty workload helps to manifest the performance difference between different algorithms, as we learnt from previous experiments for MDASA and MLBESA. However, results for other distributions such as normal distribution and exponential distribution all have the similar trends and thus are omitted here.

As shown in Figure 10, the DSR's of the algorithms do not have significant difference. However, DASA, LBESA, BE-v, and MDASA perform significantly better than MLBESA and D^{over} in terms of ABR (see figure 11). This result is consistent with that shown in Figure 8, i.e., LBESA performs better that MLBESA. Furthermore, the fact that D^{over} performs poorly has been described in Chapter 8, Section 8.5 of [13].

Finally, we study task response times under various algorithms. For the purpose of comparison, we use a metric called "Response Time Ratio" that is defined as the ratio of the actual task response time to its execution time. This definition does not depend on the execution times of particular tasks and thus enables fair comparison of large number of tasks.

In Figure 12, we show the average response time ratios of the six algorithms under the base line general Pareto distributions. As indicated in the figure, most of the response time ratios are close, especially for light to medium load conditions i.e., load no more than 0.8. However, when load increases, BE-v and D^{over} generally produce longer response times than other algorithms.

B. Task Response Times Under MDASA and MLBESA

The experimental results in Section V-A show that MDASA and MLBESA, in general, have close performance to their counterpart algorithms. We conjecture that MDASA and MLBESA make almost the same scheduling decisions as those made by DASA and LBESA at most scheduling events, respectively. Consequently, most of the task response times under MDASA and MLBESA should be close to that under DASA and LBESA, respectively.

If task response times under MDASA and MLBESA are almost the same as that under DASA and LBESA, respectively, then MDASA and MLBESA can be used as approximate response time analysis algorithms for estimating response times under DASA and LBESA. This would be a useful property of the algorithms as determining exact response times under best-effort schedulers such as DASA and LBESA is impossible without constructing entire task schedules [20]. Such schedule construction is computationally expensive due to the high cost of DASA and LBESA.

Thus, we could use MDASA and MLBESA to estimate response times under DASA and LBESA, respectively, in a much faster way as MDASA and MLBESA have low computational costs.[7]

To measure how close the task response times are under different scheduling algorithms, we define the relative error of task response times as follows:

$$RelError = \begin{cases} \dfrac{Resp - P_resp}{Resp} & Resp < \infty \text{ and } P_resp < \infty \\ 100\% & Resp = \infty \text{ and } P_resp < \infty \\ -100\% & Resp < \infty \text{ and } P_resp = \infty \\ 0 & Resp = \infty \text{ and } P_resp = \infty \end{cases}, \quad (7)$$

where $Resp$ and P_resp are the response times of a task under DASA (or LBESA) and MDASA (or MLBESA), respectively. Note that in the case of a deadline miss, the task response time becomes infinity. This is because an infeasible task is never to scheduled by schedulers such as DASA and LBESA and thus needs to be aborted. Since task response is defined as the time interval between the finishing time of a task and its arrival time, response time of an aborted task is calculated as infinity. This case is also handled by the above error definition.

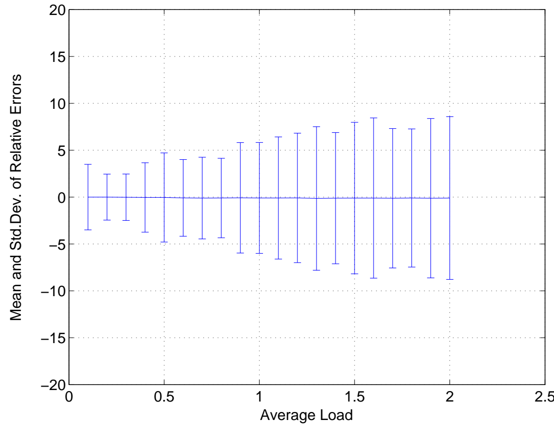

Fig. 13. Response Time Errors under MDASA

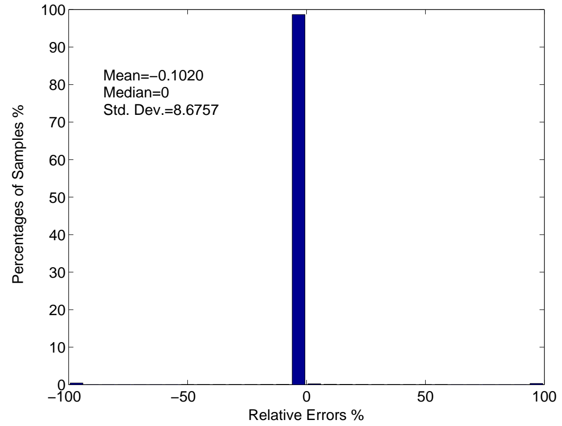

Fig. 14. Histogram of MDASA Response Time Errors

We first show the mean and standard deviation of the relative errors under MDASA in Figure 13. The experimental data are collected with the baseline exponential distribution of

[7]In fact, this has been one of our primary motivations for developing MDASA and MLBESA in the first place (as described in Section I).

task parameters shown in Table IV of Section V-A. Each error bar in the plot corresponds to the *Mean* and *StdDev* at one load point, which is centered at the *Mean* and is $2 \times StdDev$ long.

From Figure 13, we see that the *Mean*'s are always very close to zero. We also observe that the relative errors are more spread-out as the load increases, which is implied by the larger *StdDev*. Thus, there are larger, and possibly more errors, during heavy load.

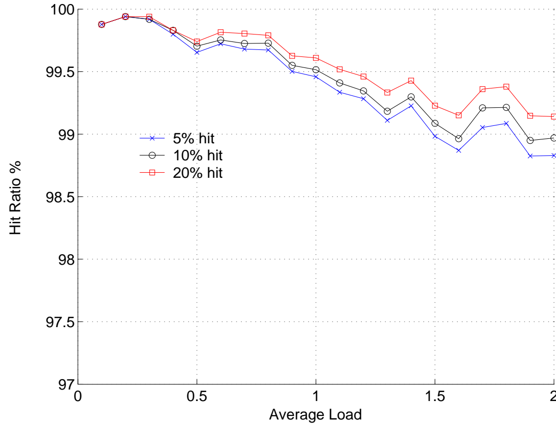

Fig. 15. Hit Ratios under MDASA

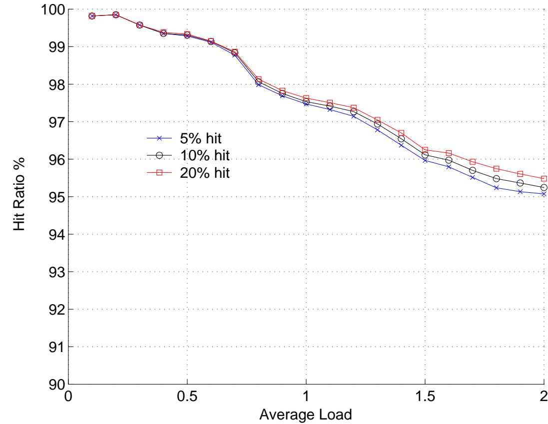

Fig. 16. Hit Ratios under MLBESA

To illustrate the distribution of the relative errors, we show a sample histogram of MDASA's response time relative errors at $\rho_A = 2.0$ (for the baseline exponential distribution experiment) in Figure 14. We observe that more than 95% of the relative errors are very close to 0. Furthermore, we see that there are a few very large errors, 100% or -100% for example.

Thus, the relatively large *StdDev* could be due to the few large errors. The error bar plot is hence misleading. For capturing the error distribution more accurately, we use an extended *outlier* measurement. We define 5% *Hit Ratio* as the ratio of the relative errors within ±5% to the total number of samples. 10% *Hit Ratio* and 20% *Hit Ratio* have similar definitions.

Figure 15 shows the 5%, 10%, and 20% hit ratios for the same baseline exponential experiment under MDASA at $\rho_A = 2.0$. Being consistent with the histogram, the hit ratios are all around 99%. This means that almost all the task response times under MDASA are very close to the values under DASA. The hit ratio plots under MLBESA are shown in Figure 16.

C. *Influences of Task Parameters on Response Times*

In this section, we study the influence of task parameters on task response times. Since one of our objectives is to use the task response times obtained under the heuristic MDASA

and MLBESA algorithms as an approximation for the response times that can be obtained under their deterministic counterparts, we are interested in determining how relative errors in response times are affected by task parameters.

We studied the influence of task parameters in two ways: (1) by determining the effect on *Hit Ratio*, defined in Section V-B and (2) by determining the correlation between task parameters and response time errors. In general, a distribution is characterized by its central tendency and dispersion, which are usually specified as $Mean$ and *StdDev*. We chose the normal distribution as the basic distribution in these experiments, because the $Mean$ and *StdDev* of a normal distribution are independent parameters. Furthermore, the normal distribution may be approximated by the Central Limit Theorem [33] in many real applications. The general Pareto distribution is also used in the experiments because it allows studying the effect of infinite variance.

In each experiment, the $Mean$ or *StdDev* of one of the four task parameters C_i, S_i, B_i, and I_i, is varied while the other parameters are kept the same, as shown in Table V of Section V-A.

1) Influence of Timing Parameters: Our first experiment compared the 10% hit ratio of MDASA for 0.5 seconds of mean slack and for 0.25 seconds of mean slack. The results are shown in Figure 17. The influence of the *StdDev* of the slack for MDASA is shown in Figure 18.

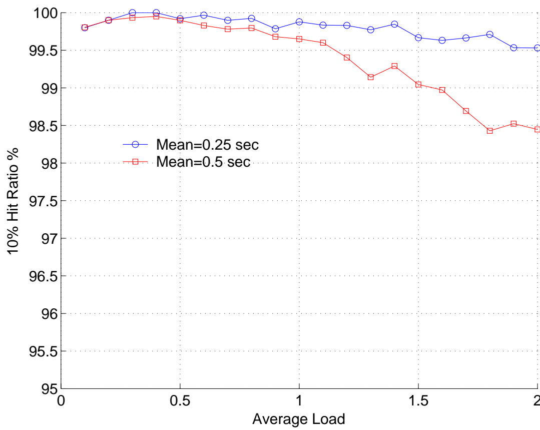

Fig. 17. Influence of Mean Slack under MDASA

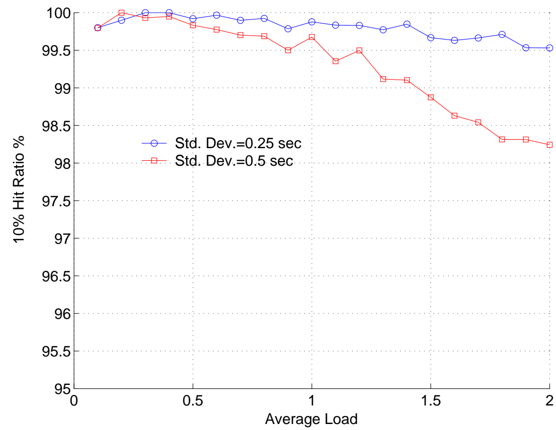

Fig. 18. Influence of StdDev Slack under MDASA

We observe that either a larger $Mean$ or a larger *StdDev* of task slack slightly degrades the performance of the algorithm. A larger *StdDev* implies that the tasks are more “bursty.” Thus, we conjecture that the length of the task ready queue may vary in a larger range. Recall that MDASA *heuristically* determines the probability that a task is in the feasible schedule

based on the queue length. Therefore, MDASA has a greater chance of making a different scheduling decision from that of DASA if there is a long ready queue. Furthermore, we use the “trucated” normal distribution i.e., only positive values are allowed, to generate the task slack, since we expect a task itself is feasible at the time of its arrival. Thus, increasing the *Mean* could also potentially generate a more bursty task stream. However, the performance degradation of the algorithm is found to be very small in both cases.

We observed that the influence of task execution times and inter-arrival times follow the similar pattern as that of the slack. That is, doubling *Mean* or doubling *StdDev* of the task execution time or inter-arrival time may result in a slightly worse performance, e.g. 1% degradation in terms of 10% hit ratio. We also observed similar results for MLBESA. For brevity, we omit these plots here.

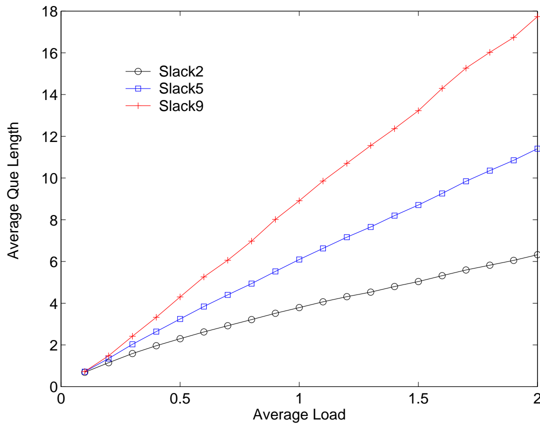

Fig. 19. Average Queue Length Under MDASA

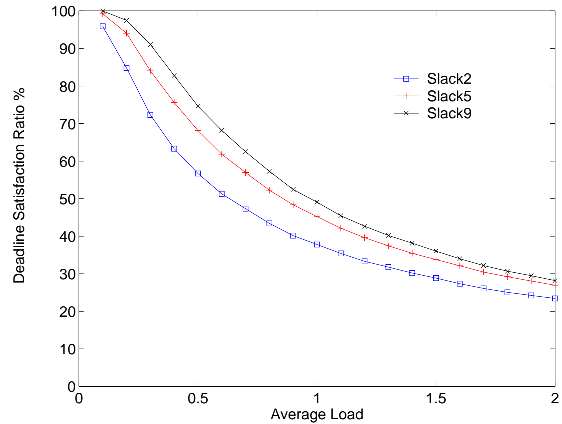

Fig. 20. Impact of Slack Times on MDASA Performance

To reenforce the results presented in Figures 17 and 18, we also conducted experiments for studying the effect of task slack times on algorithm performance. These experiments are conducted with the baseline general Pareto distribution. Furthermore, we changed the mean slack times to twice (slack2), five times (slack5), and nine times (slack9) of task execution times.

Figure 19 shows the average queue length produced by MDASA. Note that infeasible tasks are dropped from the ready queue to avoid infinite ready queue. These plots clearly show that besides workload governed by task execution times and inter arrival, task slack times have significant impact on the length of ready queues. Furthermore, in Figure 20, we show DSR’s achieved by MDASA for the base-line Pareto distribution with various of slack times. As can be seen from the figure, a large slack time implies less stringent timing constraint, and thus

yields better performance.

2) Influence of Benefit: Unlike the task timing parameters such as C_i, S_i, and I_i, the real-time benefit of a task indicates the importance level of a task *relative* to other tasks in the system. Thus, we conjecture that the performance of the algorithm is benefit scale-invariant.

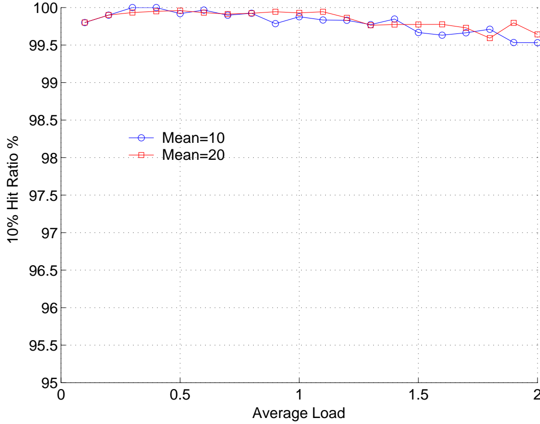

Fig. 21. Influence of Mean Benefit under MDASA

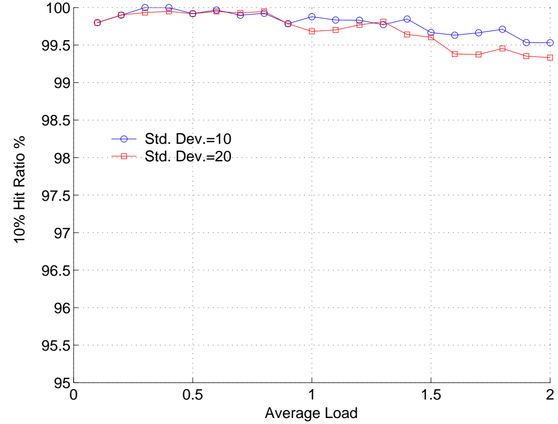

Fig. 22. Influence of StdDev Benefit under MDASA

Figure 21 and Figure 22 show the 10% hit ratio curves with different $Mean$ and $StdDev$ of real-time benefit for the MDASA algorithm. As we expect, the performance of the algorithm does not change significantly.

3) Correlation between Task Parameters and Response Time Error: We were also interested in determining whether there is any significant correlation between task parameters and response time errors. For example, does a task with longer execution time has a larger response time error than others? We consider the Pearson and Spearman correlations [33] for measuring this aspect.

Table VI and Table VII show the correlation coefficients between the task parameters and the response time relative error at $\rho_A = 0.2$ and $\rho_A = 2.0$, respectively.

TABLE VI

CORRELATIONS AT $\rho_A = 0.2$

Correlation	S_i	C_i	B_i
Pearson	0.0041	0.0518	0.0036
Spearman	0.0137	0.0371	0.0075

TABLE VII

CORRELATIONS AT $\rho_A = 2.0$

Correlation	S_i	C_i	B_i
Pearson	0.0025	0.0101	0.0335
Spearman	0.02	0.0208	0.0314

Note that the task inter-arrival time depends upon the load and the execution time of the task stream. Thus, it is not shown as a separate parameter in the tables. The Pearson

correlation measures the linear relationship between two quantities, which may be a too strict requirement in our case. But the rank-order correlation measured by Spearman correlation does not show any significance either. Therefore, we conclude that the parameters of a task do not have significant correlation with the response time difference between the MDASA and DASA algorithms. The correlation coefficients under MLBESA also lead us to the same conclusion.

VI. Related Work

Value- or benefit-based overload scheduling has been studied in different contexts [2], [6], [10], [11], [15], [21], [26], [40], [3], [7]–[9], [12], [34], [35], [38].

Baruah et. al. found the $\frac{1}{\left(1+\sqrt{k}\right)^2}$ upper bound on the competitive factor of any on-line scheduling algorithm, given the *importance ratio* of k [6]. Note that k is defined as the maximum task value density divided by the minimum task value density among the task set. This upper bound is achieved by the D^{over} algorithm presented in [26]. However, the competitive factor of an on-line algorithm only measures the *worst-case* performance of the algorithm, which may not be directly related to the *average* performance for a large number of random tasks. Furthermore, our experimental results shown in Section V-A suggest that D^{over} may perform worse than MDASA and MLBESA for random task sets, which is also described in [13].

Baruah et al. also proposed two metrics for measuring the performance of a scheduling algorithm during overload situations i.e., the "Effective Processor Utilization(EPU)" and "Completion Count(CC)" [9]. It is shown that no on-line algorithm is competitive with respect to CC during overload situations [9]. In the case of "uniform value", where the value of a task is equal with its execution time, optimal scheduling with known minimum slack factor and multiple processors are explored in [8] and [7], respectively.

Besides the optimal algorithms such as D^{over}, heuristic algorithms have been proposed. In [40], the authors considered the problem of computing schedules in a dynamic environment by ensuring the feasibility of the system. Furthermore, several algorithms similar to the LBESA algorithm have been designed [2], [12], [34]. These algorithms are similar to LBESA in that they reject tasks, by ascending order of task benefit density or variant metrics of benefit density, to resolve any overload situation.

The RED (Robust Earliest Deadline) algorithm proposed by Buttazzo and Stankovic [12] combines many features including graceful performance degradation during overload, deadline

tolerance and resource reclaiming. Since we assume that the execution time is the exact processor time demand of a task, no resource reclaiming mechanism is needed in this task model. Similarly, no deadline tolerance is allowed for rectangular benefit functions. Note that the RED algorithm itself does not specify any particular policy to reject tasks when overload occurs. In the case of rejecting tasks in ascending order of task benefit densities, the RED scheduler behaves like a LBESA scheduler. Thus, we do not directly compare the performance of MDASA and MLBESA with the RED algorithm.

Mosse et al. derived several variants of the LBESA algorithm in [34], which also showed that the BE-v variant may outperform LBESA when the value accrued by rejecting short teasers is greatest. Our simulation experiments show that MDASA, MLBESA, and BE-v have very close performance. Furthermore, a comparative study show that the robust algorithms, such as RED, performs better than the plain algorithms, such as plain EDF, and guarantee-based algorithms [11]. Therefore, we focus our performance comparison with only a class of robust algorithms, where the BE-v algorithm serves as an example.

Aldarmi and Burns proposed the concept of "timeliness-function" in [3]. Unlike the rectangular benefit function which drops at the task deadline, the timeliness-function decreases when the slack time of a task reaches zero. In [2], the authors show that scheduling the task with the highest Dynamic Timeliness-Density (DTD) is more effective than scheduling the highest benefit density task. However, the "timeliness" of a task has already been examined in the feasibility test of DASA and LBESA, which abort any task with negative slack. Thus, DTD scheduling bears a lot of similarities with the DASA algorithm. Burns et al. also studied the problem of assigning value to tasks in a systematic and rational way [10], [35].

Apart from the task model with one segment of execution per task, the concept of "imprecise computations" has also been proposed as an effective technique to handle overloads [29]. The imprecise computation model assumes that a task consists of a mandatory part that must finish execution before its deadline, and an optional part that can be executed to improve the quality of computation.

In [32], Mejia-Alvarez assumes this imprecision computation model and used combinatorial algorithms to find near-optimal solutions. The work on feedback control theory scheduling further extends the imprecise computation model and assumes the presence of N versions of the same task ($N \geq 2$) [31]. The MDASA and MLBESA algorithms do not assume the existence of multiple versions of the same task.

In [36], the author presents an overload management scheme that can satisfy deadlines of at least m instances of a periodic task within k consecutive releases, which is called the (m, k) *firm guarantee*. However, no benefit is defined for the application tasks, which fundamentally differs from our model.

In [15], [21], the authors consider the problem of scheduling a set of non-preemptive tasks to maximize the accrued benefit, which complement the work of scheduling independent tasks. Richardson et. al. consider scheduling tasks *before* overload but *after* a fault occurs [38], where a fault could be the reason of the overload situation. Our task model does not assume this in-advance knowledge and hence is more general.

Furthermore, majority of the algorithms with comparable performance to DASA and LBESA repeatedly determine the task set feasibility and hence is expensive. The proposed MDASA and MLBESA algorithms, on the other hand, heuristically determine the feasible task subset and is much faster.

Although probabilistic analysis has been used in real-time scheduling such as in the efforts [4], [19], [43], these algorithms mainly deal with scheduling real-time processes that have varying computational demand.

The original LBESA algorithm itself is the closest work to the MDASA and MLBESA algorithms. LBESA statistically "guesses" if there is an overload, based on the variance of the total process slack time. If there is an overload, LBESA removes the least benefit density process from the feasible process set that it computes, where the benefit density of a process is the process benefit divided by its remaining execution time. This procedure is repeated until the algorithm "guesses" and concludes that there is no overload. MDASA and MLBESA algorithms avoid this iterative decision procedure of LBESA by heuristically determining the processes in the *final* feasible schedule.

VII. Conclusions

In this paper, we present two fast, best-effort, real-time scheduling algorithms called MDASA and MLBESA. MDASA and MLBESA are novel in the way that they heuristically, yet accurately, mimic the behavior of the DASA and LBESA algorithms, but are faster with $O(n)$ and $O(n \log(n))$ worst-case complexities, respectively. MDASA and MLBESA reason about the behavior of DASA and LBESA by heuristically determining a feasible schedule of the task ready queue.

Our experimental results show that MDASA and MLBESA perform almost as good as DASA and LBESA, respectively, unless the workload is highly bursty and the system is heavily overloaded. Furthermore, the task response times under MDASA and MLBESA are found to be very close to the values under their counterpart scheduling algorithms. While MDASA performs better than MLBESA and has a better worst-case complexity, MLBESA guarantees the optimal schedule during underload situations. Our prototype implementation using a middleware [28] also suggests the effectiveness of MDASA/MLBESA, e.g., response time analysis using MDASA/MLBESA can be 100∼200 msec faster than that using DASA/LBESA. Thus, the major contribution of the paper is the MDASA and MLBESA algorithms.

References

[1] T. Abdelzaher, "An automated profiling subsystem for qos-aware services," in *Proc. IEEE Real-Time Technology and Applications Symposium*, 2000, pp. 208–217.

[2] S. Aldarmi and A. Burns, "Dynamic value-density for scheduling real-time systems," in *Proc. Euromicro Conference on Real-Time Systems*, June 1999, pp. 270–277.

[3] S. A. Aldarmi and A. Burns, "Time-cognizant value functions for dynamic real-time scheduling," Department of Computer Science, The University of York, U.K., Tech. Rep., 1998, YCS-306.

[4] A. Atlas and A. Bestavros, "Statistical rate monotonic scheduling," in *Proc. IEEE Real-Time Systems Symposium*, December 1998, pp. 123–132.

[5] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, and L. Rosier, "On the competitiveness of on-line task real-time task scheduling," in *Proc. IEEE Real-Time Systems Symposium*, December 1991, pp. 106–115.

[6] S. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. Rosier, and D. Shasha, "On-line scheduling in the presence of overload," in *Proc. Annual Symposium on Foundations of Computer Science*, October 1991, pp. 101–110.

[7] S. K. Baruah, "Overload tolerance for single-processor workloads," in *Proc. IEEE Real-Time Technology and Applications Symposium*, June 1998, pp. 2–11.

[8] S. K. Baruah and J. Haritsa, "Robust: a hardware solution to real-time overload," in *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, vol. 21, no. 1, 1993, pp. 207–216.

[9] S. K. Baruah, J. R. Haritsa, and N. Sharma, "On-line scheduling to maximize task completions," *The Journal of Combinatorial Mathematics and Combinatorial Computing*, vol. 39, pp. 65–78, 2001.

[10] A. Burns, D. Prasad, A. Bondavalli, F. D. Giandomenico, K. Ramamritham, J. Stankovic, and L. Strigini, "The meaning and role of value in scheduling flexible real-time systems," *Journal of Systems Architecture*, vol. 46, pp. 305–325, 2000.

[11] G. Buttazzo, M. Spuri, and F. Sensini, "Value v.s. deadline scheduling in overload conditions," in *Proc. IEEE Real-Time Systems Symposium*, 1995, pp. 90–95.

[12] G. Buttazzo and J. Stankovic, "Red: Robust earliest deadline scheduling," in *Proc. International Workshop on Responsive Computing Systems*, September 1993, pp. 100–111.

[13] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, 1997.

[14] G. C. Buttazzo and J. Stankovic, "Adding robustness in dynamic preemptive scheduling," in *Responsive Computer Systems: Steps Toward Fault-Tolerant Real-Time Systems*, D. Fussel and M. Malek, Eds. Kluwer Academic Publishers, October 1995, pp. 67–88.

[15] K. Chen and P. Muhlethaler, "A scheduling algorithm for tasks described by time value function," *Journal of Real-Time Systems*, vol. 10, no. 3, pp. 293–312, 1996.

[16] R. Clark, E. D. Jensen, A. Kanevsky, J. Maurer, P. Wallace, T. Wheeler, Y. Zhang, D. Wells, T. Lawrence, and P. Hurley, "An adaptive, distributed airborne tracking system," in *Proc. IEEE International Workshop on Parallel and Distributed Real-Time Systems*, LNCS, vol. 1586. Springer-Verlag, April 1999, pp. 353–362.

[17] R. K. Clark, "Scheduling dependent real-time activities," Ph.D. dissertation, Carnegie Mellon University, 1990, CMU-CS-90-155.

[18] M. L. Dertouzos, "Control robotics: the procedural control of physical processes," in *Proc. IFIP Congress*, 1974, pp. 807–813.

[19] M. K. Gardner, "Probabilistic analysis and scheduling of critical soft real-time systems," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 1999.

[20] T. Hegazy and B. Ravindran, "Using application benefit for proactive resource allocation in asynchronous real-time distributed system," *IEEE Transactions on Computers*, vol. 51, no. 8, pp. 945–962, August 2002.

[21] S.-I. Hwung, C.-M. Chen, and A. K. Agrawala, "Scheduling an overloaded real-time system," in *Proc. IEEE International Phoenix Conference on Computers and Communications*, June 1996, pp. 22–28.

[22] E. D. Jensen, "Asynchronous decentralized real-time computer systems," in *Real-Time Computing*, ser. Proc. NATO Advanced Study Institute, W. A. Halang and A. D. Stoyenko, Eds. Springer Verlag, October 1992.

[23] E. D. Jensen and J. D. Northcutt, "Alpha: A non-proprietary operating system for large, complex, distributed real-time systems," in *Proc. IEEE Workshop on Experimental Distributed Systems*, Huntsville, Alabama, 1990, pp. 35–41.

[24] E. D. Jensen and B. Ravindran, "Guest editor's introduction to special section on asynchronous real-time distributed systems," *IEEE Trans. on Computers*, vol. 51, no. 8, pp. 881–882, August 2002.

[25] G. Koob, "Quorum," in *Proc. Darpa ITO General PI Meeting*, October 1996, pp. A–59–A–87.

[26] G. Koren and D. Shasha, "D-over: An optimal on-line scheduling algorithm for overloaded real-time systems," in *Proc. IEEE Real-Time Systems Symposium*, December 1992, pp. 290–299.

[27] W. E. Leland, M. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic," in *Proc. ACM SIGCOM*, 1993, pp. 183–193.

[28] P. Li, B. Ravindran, J. Wang, and G. Konowicz, "Choir: A real-time middleware architecture supporting benefit-based proactive resource allocation," in *Proc. IEEE International Symposium on Object-oriented Real-time distributed Computing*, 2003, pp. 292–299.

[29] J. W. S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung, "Imprecise computations," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 83–94, Janurary 1994.

[30] C. D. Locke, "Best-effort decision making for real-time scheduling," Ph.D. dissertation, Carnegie Mellon University, 1986, CMU-CS-86-134.

[31] C. Lu, J. Stakovic, G. Tao, and S. Son, "Feedback control real-time scheduling: Framework, modeling and algorithms," *Journal of Real-Time Systems*, vol. 23, no. 1/2, pp. 85–126, July 2002.

[32] P. Mejia-Alvarez, R. Melhem, and D. Mosse, "An incremental approach to scheduling during overloads in real-time systems," in *Proc. IEEE Real-Time Systems Symposium*, December 2000, pp. 283–293.

[33] W. Mendenhall and T. Sincich, *Statistics for the Engineering and Computer Sciences*, 3rd ed. Dellen Pub. Co., 1991.

[34] D. Mosse, M. E. Pollack, and Y. Ronen, "Value-density algorithm to handle transient overloads in scheduling," in *Proc. Euromicro Conference on Real-Time Systems*, June 1999, pp. 278–286.

[35] D. Prasad and A. Burns, "A value-based scheduling approach for real-time autonomous vehicle control," *Robotica*, vol. 31, pp. 273–279, 2000.

[36] P. Ramanathan, "Overload management in real-time control applications using the (m, k) firm guarantee," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 6, pp. 549–559, 1999.

[37] B. Ravindran, "Engineering dynamic real-time distributed systems: Architecture, system description language, and middleware," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 30–57, January 2002.

[38] P. Richardson and S. Sarkar, "Adaptive scheduling: Overload scheduling for mission critical systems," in *Proc. IEEE Real-Time Technology and Applications Symposium*, June 1999, pp. 14–23.

[39] B. Shirazi, L. R. Welch, B. Ravindran, C. Cavanaugh, and E. Huh, "Dynbench: A benchmark suite for dynamic real-time systems," *Journal of Parallel and Distributed Computing Practices*, vol. 3, no. 1, pp. 89–107, March 2000.

[40] J. A. Stankovic and K. Ramamritham, "The spring kernel: A new paradigm for real-time systems," *IEEE Software*, vol. 8, no. 3, pp. 62–72, May 1991.

[41] D. Stewart and P. Khosla, "Real-time scheduling of sensor-based control systems," in *IFAC/IFIP Workshop Real-Time Programming*, May 1991, pp. 144–150.

[42] The Open Group, *MK7.3a Release Notes.* Cambridge, Massachusetts: The Open Group Research Institute, October 1998.

[43] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W.-S. Liu, "Probabilistic performance guarantee for real-time tasks with varying computation times," in *Proc. IEEE Real-Time Technology and Applications Symposium*, 1995, pp. 164–173.

[44] D. von Seggern, *CRC Standard Curves and Surfaces.* CRC Press, 1993, pp.252.

[45] C. A. Waldspurger and W. E. Weihl, "Lottery scheduling: Flexible proportional-share resource management," in *Operating Systems Design and Implementation*, 1994, pp. 1–11.

[46] L. R. Welch, B. Ravindran, B. Shirazi, and C. Bruggeman, "Specification and modeling of dynamic, distributed real-time systems," in *Proc. IEEE Real-Time Systems Symposium*, December 1998, pp. 72–81.

[47] L. R. Welch and B. A. Shirazi, "A dynamic real-time benchmark for assessment of qos and resource management technology," in *Proc. IEEE Real-Time Technology and Applications Symposium*, June 1999.